NDDL Parser and Compiler

The NDDL Parser and Compiler

The NDDL module encompasses a couple of standalone tools for parsing NDDL and compiling the resulting XML into C++ code. The module also includes domain-independent C++ code which is used by the generated code.

Invoking the Parser and Compiler

When creating a project using the makeproject tool, it is not necessary to invoke the NDDL parser directly as the generated makefile will invoke it. In most other circumstances, it becomes necessary to understand the process of configuring and executing the NDDL parser.

Given nddl.jar (which will be generated in $EUROPA_HOME/lib when you build EUROPA), execute `java -jar nddl.jar` *filename.nddl* which will run both the parser and code generator on *filename.nddl*. Output files will be named *filename.xml*, *filename.hh*, and *filename.cc* and will be in the same directory as *filename.nddl*. To suppress code generation (e.g. to use interpreted NDDL or generate an initial state file) pass the jarfile the --NddlParser option prior to the filename. For further options run `java -jar nddl.jar --help`.

When the NDDL parser is started it will attempt to find its configuration file, NDDL.cfg. It will first look in the current directory; if the file is not found it will look in the $EUROPA_HOME directory, and then in the $PLASMA_HOME directory (if defined).

A: The NDDL Parser

The NDDL Parser transforms NDDL into XML. It performs some analysis and semantic checking during the transformation. It is available as a standalone tool.

The NDDL Parser is based on ANTLR. There are two passes to the parse process. The first pass parses the input NDDL input an abstract syntax tree. The second pass parses the abstract syntax tree into an in-memory XML structure. This structure is passed to a class which processes it and determines type information. Finally, the XML is passed to a semantic checker, before being written out. The XML is manipulated using NanoXML.

The NDDL Parser has a command line interface which is built as part of nddl.jar when the NDDL module is built. Generally the parser is invoked to parse a single NDDL file into xml. An example of how to invoke the parser is the $EUROPA_HOME/dist/europa/bin/nddlparse executuable which looks similar to this:

```
nddlparse -o filename.xml filename.nddl
```

In this example, the program looks for a file called *filename.nddl* in the current directory and passes it to the NDDL parsing routines. After parsing, adding types, and checking the semantics, the program writes the resulting nddl-xml to a file called "filename.xml" in the current directory.

NOTE: The parser requires the third-party libraries antlr.jar and nanoxml.jar which are packaged with EUROPA.

B: The NDDL Compiler

NDDL can be interpreted by calling the method PSEngine::executeScript, and specifying as the language either "nddl", or "nddl-xml" (which is the output from step A above).

Alternatively, you can generate C++ code for the model and link it into your project by using the NDDL compiler.

The NDDL Compiler transforms nddl-xml into C++ code. It performs a semantic check on the XML and then generates a header file and an implementation file. It is available as a standalone tool.

First, the NDDL Compiler reads in the XML for all input files. Next, it passes the XML through the semantic checker. If it passes the semantic check, it processes the input using the header generator to write the header (.hh) file. Finally, it processes the input using the implementation generator to write the implementation (.cc) file.

The NDDL Compiler makes use of a number of classes for writing specific constructs, such as enumerations, predicates, rules. It also uses several utility classes that provide services such as indented printing and common xml traversals for models.

The NDDL Compiler has a command line interface which is built as part of nddl.jar when the NDDL module is built. Generally the compiler is invoked to parse a single XML file into a header file and implementation file. An example of how to invoke the compiler is the script $EUROPA_HOME/dist/europa/bin/nddlcompile which looks similar to this:

```
nddlcompile -o filename.cc filename.xml
```

In the above example, the program looks for a file called "filename.xml" in the current directory and passes it to the XML loading routines. After loading and checking the semantics of the XML, it generates the header and then the body, in files called "filename.hh" and "filename.cc", which are both written to the current directory.

NOTE: The compiler also requires the third-party library nanoxml.jar (packaged with EUROPA).

NDDL Support Code

Aside from the above two tools the NDDL module wraps up some additional NDDL support code. This is mostly to simplify the work of the compiler by predefining some macros and other utilities. Also important are these nddl files that may need to be included in your model to get some built-in NDDL functionality:

- PlannerConfig.nddl
- Plasma.nddl
- Resources.nddl